A Local Corrections Algorithm for Solving Poisson's Equation in Three Dimensions

Peter McCorquodale* Phillip Colella
Lawrence Berkeley National Laboratory
Gregory T. Balls Scott B. Baden
University of California, San Diego

October 30, 2006

Abstract

We present a second-order accurate algorithm for solving the free-space Poisson's equation on a locally-refined nested grid hierarchy in three dimensions. Our approach is based on linear superposition of local convolutions of localized charge distributions, with the nonlocal coupling represented on coarser grids. The representation of the nonlocal coupling on the local solutions is based on Anderson's Method of Local Corrections and does not require iteration between different resolutions. A distributed-memory parallel implementation of this method is observed to have a computational cost per grid point less than three times that of a standard FFT-based method on a uniform grid of the same resolution, and scales well up to 1024 processors.

1 Introduction

We want to compute the solution to Poisson's equation on $\mathbb{R}^3$ with a charge distribution ρ with support on a compact set Ω. Specifically, we seek the solution ϕ to

$$\Delta\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = \rho(x,y,z) \tag{1}$$

that has the far-field behavior

$$\phi(\boldsymbol{x}) = -\frac{Q}{4\pi|\boldsymbol{x}|} + o\left(\frac{1}{|\boldsymbol{x}|}\right), \qquad |\boldsymbol{x}| \to \infty; \tag{2}$$

$$Q = \int_\Omega \rho(\boldsymbol{x}) d\boldsymbol{x}. \tag{3}$$

---

[*] Corresponding author. Tel. +1-510-495-2458; fax +1-510-495-2505. *E-mail address:* `PWMcCorquodale@lbl.gov`

Using the maximum principle for harmonic functions, it is not difficult to show that equations (1)–(2) have a unique solution. This solution can be written as a convolution with the Green's function G [16]:

$$\phi(\boldsymbol{x}) = (G * \rho)(\boldsymbol{x}) \equiv \int G(\boldsymbol{x}-\boldsymbol{y})\rho(\boldsymbol{y})\, d\boldsymbol{y}\ ,\ G(\boldsymbol{z}) = -\frac{1}{4\pi|\boldsymbol{z}|}. \tag{4}$$

Solutions to (1) have a strong form of elliptic local regularity. If $D \subset \Omega$ is contained in a ball of radius r, then the function

$$\phi_D(\boldsymbol{x}) = \int\limits_D G(\boldsymbol{x}-\boldsymbol{y})\rho(\boldsymbol{y}) d\boldsymbol{y} \tag{5}$$

is real analytic at all points not contained in D, and its derivatives are rapidly decaying functions of $dist(\boldsymbol{x}, D)/r$. This suggests that an efficient method for computing the potential ϕ would be to compute local convolutions of the form (5) on a disjoint union of patches, and then compute the smooth global coupling among the patches using a calculation with a much coarser (and less computationally expensive) discretization. In fact, this is the underlying approach to all $O(N) - O(NlogN)$ methods for potential theory, include the Fast Multipole Method (FMM) [12] and the Method of Local Corrections (MLC) [2] for particles, and FFT-based methods [13], multigrid [7] and domain decomposition [20] for gridded data.

In principle, the same strategy should also lead to efficient parallel methods. The local convolutions are independent, and therefore can be performed in parallel on separate processors, while the nonlocal coupling between patches is representable by such a small number of degrees of freedom so as to have a negligible impact on the computational cost. For the particle methods such as FMM and MLC, this is indeed the case [3]. For algorithms for gridded data, particularly on structured and locally-structured grids, the results are mixed. FFT-based methods are probably optimal in terms of the number of floating point operations required, but are limited to uniform grids and require some form of global communication of all the data (such as transpose) or complex mappings of data onto processors. Multigrid iteration is applicable to locally-structured multiresolution grids [4, 1] and effectively exploits local regularity to reduce the number of floating point operations to a few hundred per grid point. However, it has an unacceptably high communication cost, with communication / synchronization steps required after each local relaxation step — that is, every few tens of floating point operations per grid point. Furthermore, there is so little computation being done between communication steps that the opportunity to overlap computation with communication is limited. The domain decomposition methods have typically led to iterative methods by constructing a dense linear system for the degrees of freedom on the boundaries between subdomains using a Schur complement. Such approaches reduce that communication load somewhat, but are still iterative, and for Poisson's equation are substantially more compute-intensive than multigrid or FFT-based methods.

A natural strategy is to apply the ideas developed for particle methods to gridded data. For FMM, this has been done in two dimensions [9, 8] by applying the fast multipole method directly to volume potentials on the grid, with methods that have a computational cost per grid point of less than three times that of an FFT on a uniform grid, and furthermore have the locality of the FMM approach with respect to communication. However, the direct extension of that approach to three dimensions, while feasible, will not have the same absolute floating-point performance of a modest integer multiple of that of an FFT-based method, due to the substantially larger cost per grid point of the FMM method for computing volume potentials in 3D relative to to that of 2D. To deal with that problem, one can take the approach of Greengard and Lee [11], in which local volume potentials on patches are computed using fast transform methods, with the FMM at the boundaries of patches to resolve the mismatch in the solutions at patch boundaries as well as the nonlocal coupling between patches. Using FMM only on two-dimensional surfaces might reduce the cost of that part of the calculation so as to make the overall floating-point cost, relative to FFT, more like that of the 2D FMM-based algorithms. However, such an approach has been carried out to date only in 2D.

The starting point for our approach is an extension of Anderson's MLC algorithm in two dimensions to gridded data in two dimensions [5, 6]. In this approach, local convolutions are computed using the James–Lackner method [14, 17] of representing infinite-domain boundary condition in terms of solutions to two Dirichlet problems on nested domains, plus a boundary-to-boundary convolution. The nonlocal coupling between patches is represented by solving a coarse grid problem and interpolating a correction back to the fine grid patches in a manner similar to full-approximation-storage multigrid. Unlike multigrid, though, the method is noniterative. In the present work, we generalize the method to locally-refined grids in three dimensions. A principal technical issue is the generalization to three dimensions of the James–Lackner method for computing local convolutions. We do this using a simplified FMM to compute the boundary-boundary convolutions, combined with FFT methods to compute the volume potentials. Thus, the method is similar in spirit to the approach of Greengard and Lee [11], but with different technical details.

2 Preliminaries

We represent both the potential field, ϕ, and the charge, ρ, on a discrete, three-dimensional Cartesian grid, with grid points spaced equally in all three directions by the same mesh spacing h. A triple of integers $\boldsymbol{i} = (i_x, i_y, i_z)$ indexes a point in real space $\boldsymbol{x_i} = (i_x h, i_y h, i_z h)$. Typically, our computational domain will be described in terms of unions of rectangular patches of the form $\Omega^h = [\boldsymbol{l}, \boldsymbol{u}]$, where $\boldsymbol{l}$ and $\boldsymbol{u}$ are the integer triples corresponding to the lower and upper corners of the region. Our grids are node-centered, with Ω^h representing the region in physical space $[\boldsymbol{l}h, \boldsymbol{u}h]$. Thus a union of rectangular patches representing a disjoint union of regions in physical space may have non-empty intersections in index space. We define a coarsening operator $\mathcal{C}$ as

$$\mathcal{C}(\Omega^h, C) = [\lfloor \boldsymbol{l}/C \rfloor, \lceil \boldsymbol{u}/C \rceil] \tag{6}$$

where the operators $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ round up and down to the nearest integer, respectively. We also need the *grow* operation $\mathcal{G}$, which extends or shrinks an index domain by a uniform amount in each direction:

$$\mathcal{G}(\Omega^h, p) = [\boldsymbol{l} - (p,p,p), \boldsymbol{u} + (p,p,p)]. \tag{7}$$

When $p < 0$, $\mathcal{G}$ returns a shrunken domain. We denote by $\partial\Omega^h$ the set of boundary points of Ω^h:

$$\partial\Omega^h = \Omega^h - \mathcal{G}(\Omega^h, -1). \tag{8}$$

A field ψ is represented on this discrete grid by ψ^h such that

$$\psi^h_{\boldsymbol{i}} \approx \psi(\boldsymbol{x_i}). \tag{9}$$

We can also define a sampling operator $\mathcal{S}$ that projects a discrete field on Ω^h onto $\mathcal{C}(\Omega^h, C)$:

$$\mathcal{S}(\psi^h, C)_{\boldsymbol{i}} = \psi^h_{C\boldsymbol{i}} \text{ , } \boldsymbol{i} \in \mathcal{C}(\Omega^h, C).$$

We denote by χ^h a discrete characteristic function defined as follows. For an interval $[l, u]$ with l and u integers, the function $\chi^h_{[l,u]}$ on the real line has the value 1 in the interval (l, u), 0 outside $[l, u]$, and $\frac{1}{2}$ at l and u. Then for a box $B = [\boldsymbol{l}, \boldsymbol{u}]$, the function χ^h_B is defined on index space as the product of interval functions over all dimensions: $\chi^h_B = \chi^h_{[\mathtt{l}_x,\mathtt{u}_x]}\chi^h_{[\mathtt{l}_y,\mathtt{u}_y]}\chi^h_{[\mathtt{l}_z,\mathtt{u}_z]}$. We also define a characteristic function χ for the corresponding region in $\mathbb{R}^3$,

$$\chi_D(\boldsymbol{x}) \quad = \quad \begin{cases} 1, & \text{if } \boldsymbol{x} \in [\boldsymbol{l}h, \boldsymbol{u}h] \text{ ;} \\ 0, & \text{otherwise.} \end{cases}$$

We use a discretization of the Laplacian operators with the stencil points contained in a three-by-three block surrounding the evaluation point:

$$(\Delta^h \phi^h)_{\boldsymbol{i}} = \frac{1}{h^2} \sum_{\boldsymbol{j} \in \{-1,0,1\}^3} a_{||\boldsymbol{j}||} \phi^h_{\boldsymbol{i}+\boldsymbol{j}} \tag{10}$$

where $||\boldsymbol{j}||$ is the number of nonzero components in $\boldsymbol{j} \in \{-1, 0, 1\}^3$ and falls in the range $\{0, 1, 2, 3\}$. We shall use the 19-point Mehrstellen operator, specified by $a_0 = -4$, $a_1 = \frac{1}{3}$, $a_2 = \frac{1}{6}$, $a_3 = 0$. If $\phi^{exact,h}$ is the exact solution evaluated at grid points, and the truncation error, $\tau^h_{\boldsymbol{j}}$, is defined as

$$\tau^h_{\boldsymbol{j}} = \rho^h_{\boldsymbol{j}} - (\Delta^h \phi^{exact,h})_{\boldsymbol{j}}, \tag{11}$$

then we can use Taylor expansion, along with the fact that $\Delta^2\phi = \Delta\rho$, to determine that

$$\tau^h_{\boldsymbol{j}} = \rho^h_{\boldsymbol{j}} - (\Delta^h \phi^{exact,h})_{\boldsymbol{j}} = \frac{-h^2}{12}\Delta\rho + O(h^4). \tag{12}$$

Thus a solution to the system

$$\Delta^h \phi^h = \rho^h \text{ , } (\rho^h)_{\boldsymbol{j}} = \rho(\boldsymbol{j}h) \tag{13}$$

is second-order accurate: $\phi^h_{\boldsymbol{j}} = \phi^{exact,h}_{\boldsymbol{j}} + O(h^2)$. The particular form of the truncation error in (12) leads to a strong localization of the $O(h^2)$ error: if $\boldsymbol{j}h$ is contained in the complement of the closure of the support of ρ, then it is not difficult to show that $\phi^h_{\boldsymbol{j}} = \phi^{exact,h}_{\boldsymbol{j}} + O(h^4)$ [6]. More classically, one can also precondition the charge and solve

$$\Delta^h \phi^{*,h} = \rho^{*,h} = \rho^h + \frac{h^2}{12}\widetilde{\Delta}^h \rho^h, \tag{14}$$

where $\widetilde{\Delta}^h$ is any second-order accurate discretization of the Laplacian, to obtain a solution that is $O(h^4)$ everywhere. With infinite-domain boundary conditions, it is also possible to make a Mehrstellen correction to the solution *after* solving (13):

$$\phi^h := \phi^h + \frac{h^2}{12}\rho^h. \tag{15}$$

3 Convolutions on Bounded Domains

A basic component of our method of local corrections is a single-grid solver for Poisson's equation with infinite-domain boundary conditions. We follow the approach used for the 2D problem by James [14] and Lackner [17].

Let Ω be the support of the right-hand side ρ in (1). Clearly, we can represent the solution to (1)–(2) on Ω in terms of solutions of Poisson's equation with Dirichlet boundary conditions on a slightly larger domain, where the boundary conditions are computed using the convolution operator (4). We can reduce the convolution to a boundary-boundary convolution by solving an additional Dirichlet problem. Let Ω_1 and Ω_2 contain Ω with $\Omega_2 \supset \Omega_1 \supset \Omega$. Let ϕ^1 be the solution to

$$\Delta\phi^1 = \rho \text{ on } \Omega_1 \; ; \; \phi^1 = 0 \text{ on } \partial\Omega_1$$

and define a boundary charge distribution q

$$q \equiv \frac{\partial\phi^1}{\partial\boldsymbol{n}} \text{ on } \partial\Omega_1$$

where $\boldsymbol{n}$ is the unit outward normal. Then the boundary potential ϕ_B induced by q

$$\phi_B(\boldsymbol{x}) = \int_{\partial\Omega_1} G(\boldsymbol{x}-\boldsymbol{y})q(\boldsymbol{y})\,dA_{\boldsymbol{y}} \tag{16}$$

is a solution to Laplace's equation on $\mathbb{R}^3 - \partial\Omega_1$, and satisfies the jump relations $[\phi_B] = 0$, $[\frac{\partial\phi_B}{\partial\boldsymbol{n}}] = -q$ on $\partial\Omega_1$. Thus the function ϕ given by

$$\phi \;=\; \begin{cases} \phi_1 + \phi_B, & \text{on } \Omega_1; \\ \phi_B & \text{elsewhere} \end{cases}$$

is a solution to (1)–(2). In particular, ϕ is a solution to the Dirichlet problem

$$\Delta\phi = \rho \text{ on } \Omega_2 \; ; \; \phi = \phi_B \text{ on } \partial\Omega_2$$

for any $\Omega_2 \supset \Omega_1$. Note that the calculation of the Dirichlet boundary conditions requires only the convolution of the Green's function with the boundary charge q.

We use the representation described above to compute an approximation of the convolution (4). We assume that Ω is a cube, which we discretize to obtain the discrete domain Ω^h with mesh spacing h and containing $(N+1)^3$ points. We also define discrete domains $\Omega_1^h = \mathcal{G}(\Omega^h, s_1)$ and $\Omega_2^h = \mathcal{G}(\Omega^h, s_1+s_2)$, for some $s_1, s_2 \geq 0$. Then the James–Lackner algorithm in three dimensions can be described as follows.

3D JAMES–LACKNER ALGORITHM

1. Solve the homogeneous Dirichlet problem on Ω_1:

   $$\Delta^h\phi_1^h = \rho \text{ on } \mathcal{G}(\Omega_1^h, -1) \; ; \; \phi_1^h = 0 \text{ on } \partial\Omega_1^h$$

   and compute the discrete boundary charge $q_{\boldsymbol{i}} = D_B(\phi_1^h)_{\boldsymbol{i}}$, $\boldsymbol{i} \in \partial\Omega_1^h$. We use a fourth-order one-sided difference approximation of the normal derivative for D_B, e.g.,

   $$D_B(f)_{0,j,k} = \frac{-25f_{0,j,k} + 48f_{1,j,k} - 36f_{2,j,k} + 16f_{3,j,k} - 3f_{4,j,k}}{12h}.$$

2. Given the discrete charge distribution q on $\partial\Omega_1$, compute an approximation to the convolution integral (16) to obtain $g_{\boldsymbol{i}} \approx \phi_B(\boldsymbol{i}h)$ for $\boldsymbol{i} \in \partial\Omega_2^h$.

3. Solve the inhomogeneous Dirichlet problem on Ω_2:

$$\Delta^h \phi^h = \rho \text{ on } \mathcal{G}(\Omega_2^h, -1)\ ;\ \phi^h = g \text{ on } \partial\Omega_2^h.$$

The solution of the Dirichlet problems in steps 1 and 3 can be done in $O(N^3 logN)$ operations using a fast discrete sine transform to diagonalize Δ^h. Step 2 is performed using a fast multipole method that takes advantage of the fact that the charge is defined on a union of planar surfaces:

$$\partial\Omega_1 = \Omega_1(+,0) \cup \Omega_1(-,0) \cup \Omega_1(+,1) \cup \Omega_1(-,1) \cup \Omega_1(+,2) \cup \Omega_1(-,2)$$

where $\Omega_1(+,d)$ and $\Omega_1(-,d)$ are respectively the high and low faces of Ω_1 in which coordinate $d \in \{0,1,2\}$ is fixed. Then the integral in (16) can be split up as

$$\phi_B(\boldsymbol{x}) = \Phi^{+,0}(\boldsymbol{x}) + \Phi^{-,0}(\boldsymbol{x}) + \Phi^{+,1}(\boldsymbol{x}) + \Phi^{-,1}(\boldsymbol{x}) + \Phi^{+,2}(\boldsymbol{x}) + \Phi^{-,2}(\boldsymbol{x}) \tag{17}$$

where $\Phi^{\pm,d}$ is the contribution from face $\Omega_1(\pm,d)$:

$$\Phi^{\pm,d}(\boldsymbol{x}) = \int_{\Omega_1(\pm,d)} G(\boldsymbol{x}-\boldsymbol{y}) q(\boldsymbol{y})\, dA_{\boldsymbol{y}}, \tag{18}$$

which is a planar integral. Step 2, then, can be broken down as follows.

2a. Split each face $\Omega_1^h(\pm,d)$ into patches of dimensions $r \times r$ centered at points on the face coarsened by r, where r is divisible by 4. Then calculate the multipole moments up to order M of q^h on each patch. For the patch on the face $\Omega_1^h(-,2)$ that is centered at the point $(i_0, i_1, -s_1/r)$ in r-coarsened coordinates, the (p_0,p_1) moment is

$$\begin{aligned} A_{i_0,i_1}^{p_0,p_1,-,2} &= \sum_{-r/2\le j_0\le r/2}\ \sum_{-r/2\le j_1\le r/2} w_{j_0} w_{j_1} q_{(ri_0+j_0, ri_1+j_1, -s_1)} (j_0 h)^{p_0} (j_1 h)^{p_1} \\ & \quad (0 \le p_0 + p_1 \le M; \quad p_0, p_1 \ge 0) \end{aligned} \tag{19}$$

where the w_j are the weights from Boole's rule of integration, which is $O(h^6)$ accurate:

$$w_j = \begin{cases} \frac{14}{45} & \text{if } |j| = \frac{r}{2}; \\ \frac{28}{45} & \text{if } \frac{r}{2} + j \equiv 0 \mod 4 \text{ and } |j| < \frac{r}{2}; \\ \frac{64}{45} & \text{if } j \text{ is odd;} \\ \frac{8}{15} & \text{if } \frac{r}{2} + j \equiv 2 \mod 4. \end{cases} \tag{20}$$

The moments for the other faces are computed analogously.

2b. On each face of $\partial\Omega_2^h$ coarsened by r in each dimension, plus a layer of coarse points of width P, add up the evaluations $\Phi^{\pm,d}$ of multipole expansions due to all patches of all faces of $\partial\Omega_1^h$. As an example,

$$\begin{aligned} \Phi^{-,2}(\vec{x}) = \ & \textstyle\sum_{i_0,i_1} \sum_{p_0,p_1} \ A_{i_0,i_1}^{p_0,p_1,-,2} \times \\ & \frac{(-1)^{p_0+p_1}}{p_0! p_1!} \frac{\partial^{p_0+p_1} G}{\partial z_0^{p_0} \partial z_1^{p_1}} (x_0 - i_0 r h, x_1 - i_1 r h, x_2 + s_1 h) \end{aligned} \tag{21}$$

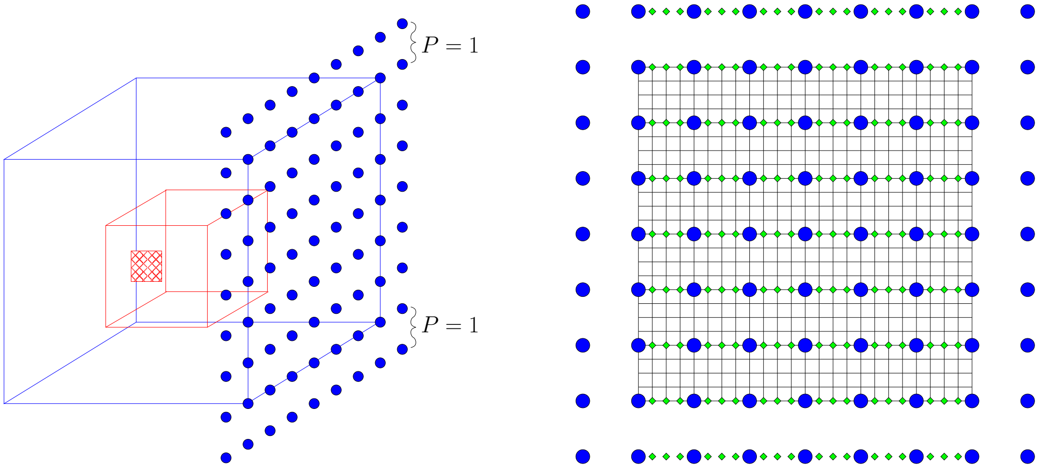

Figure 1: In step 2 of our implementation of the 3D James–Lackner algorithm, multipole moments are calculated for each patch on each face of $\partial\Omega_1^h$, such as the patch shown cross-hatched in red. The multipole expansions are then evaluated at the coarse points on the faces of Ω_2^h augmented by an additional layer of width P, indicated with blue circles for one face. These evaluations are interpolated to all the fine points on the faces of $\partial\Omega_2^h$, located at intersections of the black lines, using two passes. The evaluation points of the first pass are shown as small green diamonds.

using two-dimensional Taylor expansions of the Green's function G around the points $(x_0 - i_0rh, x_1 - i_1rh, x_2 + s_1h)$. The indices i_0, i_1 in the sum are over indices of coarse points on the face $\Omega_1^h(-, 2)$, and $p_0, p_1 \geq 0; p_0 + p_1 \leq M$.

2c. On each face of $\partial\Omega_2^h$, interpolate from the coarse points to the remaining fine points on the face, using a tensor product of Lagrange interpolating polynomials as illustrated in Figure 1.

Choosing $r \approx \sqrt{N}$ provides sufficient accuracy for the solution and allows the integration step to be completed in $O((M^2 + P)N^2)$ work. For $O(h^4)$ error, we set $M = 7$ and $P = 3$, and these are independent of N. Hence step 2 requires $O(N^2)$ work.

We also note constraints required on s_2, the spacing between Ω_1^h and Ω_2^h. Convergence requirements of the multipole method force us to choose s_2 with care. In order for the multipole expansions from a patch to converge, the distance from the center of a patch on a face of Ω_1^h to the points on the faces of Ω_2^h, on which the expansion is evaluated, should be at least twice the radius of the patch. Here we define the radius of a patch as the maximum distance from the patch center to any point on the patch. Recall that we chose our patches to be $r \times r$ fine grid points. Thus our patches have a radius of $rh/\sqrt{2}$, and the distance requirement becomes $s_2h \geq 2rh/\sqrt{2}$. We also need the number of cells along the length of Ω_2^h to be divisible by r. Combining these two requirements, we arrive at the following formula for s_2:

$$s_2 = \frac{r}{2}\left\lceil 2\sqrt{2} + \frac{N + 2s_1}{r}\right\rceil - \frac{N + 2s_1}{2}. \tag{22}$$

For efficiency, both s_1 and s_2 should be as small as possible. If the distance of the support of ρ to $\partial\Omega_1$ is nonzero, then we can set $s_1 = 0$.

We now examine the computational costs in the single-grid solver, listing the operation counts for each step:

1. FFT-based Poisson solver on Ω_1^h: $O(N^3 \log N)$.

   Normal derivatives on faces of Ω_1^h: $O(N^2)$.

2. Integration to boundary conditions on faces of Ω_2^h using FMM: $O(N^2)$.

3. FFT-based Poisson solver on Ω_2^h: $O(N^3 \log N)$.

Thus the single-grid infinite-domain solver operation count is bounded by the fast Poisson solves that use Dirichlet boundary conditions, and the overall computational cost of an infinite-domain solution is $O(N^3 \log N)$.

4 Method of Local Corrections

The domain decomposition algorithm described here is the finite-difference analogue [6] of Anderson's Method of Local Corrections (MLC) [2], extended to locally-refined nested grids in three dimensions. To simplify the presentation, we describe the MLC algorithm on two levels. We use a fine-grid discretization Ω^h corresponding to a rectangular domain Ω that contains the support of the charge ρ. Within Ω^h we have a set of cubic patches Ω_k^h of equal size that overlap only at patch boundaries. These subdomains make up a region on which the charge is finely resolved. For each patch, the charge ρ_k^h is defined on Ω_k^h. Our method entails solving local problems on each of the Ω_k^h in parallel, as well as on a single coarse global mesh Ω^H. The spacing of the coarse mesh is $H = Ch$, where C is a specified *coarsening factor*.

Because our meshes are node-centered, the points of Ω^H map directly onto corresponding points in Ω^h, and no averaging is required to coarsen the mesh data. Thus, we can coarsen the mesh by sampling the mesh without having to interpolate. In particular, we coarsen a fine grid representation using the *sample* operator $\mathcal{S}^H$: for each point $\boldsymbol{x}_C$, we can find the coarse grid value $\psi^H(\boldsymbol{x}_C)$ (where ψ^H has grid spacing H) by finding the fine grid point $\boldsymbol{x}$ at the corresponding position in ψ^h (with grid spacing $h = H/C$):

$$\psi^H(\boldsymbol{x}_C) = (\mathcal{S}^H(\psi^h))(\boldsymbol{x}/C) = \psi^h(\boldsymbol{x}). \tag{23}$$

If $\rho = \rho(\boldsymbol{x})$ is the continuous charge, we set the discrete coarse-level charge ρ^H on Ω^H and fine-level charge ρ_k^h on each Ω_k^h to be

$$\begin{aligned} (\rho^H)_{\boldsymbol{i}} &= \rho(\boldsymbol{i}H)\ \text{,}\ \boldsymbol{i} \in \Omega^H; \\ (\rho_k^h)_{\boldsymbol{i}} &= (\chi^h_{\Omega_k^h})_{\boldsymbol{i}}\, \rho(\boldsymbol{i}h)\ \text{,}\ \boldsymbol{i} \in \Omega_k^h. \end{aligned}$$

The algorithm has three computational steps interspersed by two communication steps.
METHOD OF LOCAL CORRECTIONS

1. INITIAL LOCAL SOLUTION. Using the 3D James–Lackner algorithm, calculate a local infinite-domain solution on each local subdomain, Ω_k^h, augmented with an overlap region:

$$\Delta^h \phi_k^{h,initial} = \rho_k^h \text{ on } \mathcal{G}(\Omega_k^h, s + Cb) \tag{24}$$

and construct a coarsened version of the solution, $\phi_k^{H,initial}$, by sampling:

$$\phi_k^{H,initial} = \mathcal{S}^H(\phi_k^{h,initial}) \text{ on } \mathcal{G}(\Omega_k^H, s/C + b). \tag{25}$$

Here s is a correction radius, C is the coarsening factor, and b is the width of a layer for polynomial interpolation to be used in step 3.

2. GLOBAL COARSE SOLUTION. Couple the individual local solutions by solving another Poisson equation on a coarsened mesh covering the entire domain. First construct coarsened local charge fields:

$$R_k^H = \begin{cases} \Delta^H \phi_k^{H,initial} & \text{on } \mathcal{G}(\Omega_k^H, s/C - 1); \\ 0 & \text{elsewhere} \end{cases} \tag{26}$$

and sum these charge fields to form a global coarse representation of the charge:

$$R^H = \sum_k R_k^H. \tag{27}$$

Then solve

$$\Delta^H \phi^H = R^H + (1 - \sum_k \chi_{\Omega_k^H})\rho^H \text{ on } \mathcal{G}(\Omega^H, s/C + b). \tag{28}$$

with infinite-domain boundary conditions, using the 3D James–Lackner algorithm. For this solve, we take the base domain to be $\mathcal{G}(\Omega^H, 2\lceil (s/C - 1)/2 \rceil)$ because the length must be divisible by 4.

3. FINAL LOCAL SOLUTION. Solve

$$\Delta^h \phi_k^h = \rho_k^h \text{ on } \Omega_k^h \tag{29}$$

with Dirichlet boundary conditions on $\partial\Omega_k^h$:

$$\phi_k^h(\boldsymbol{x}) = \sum_{k' : \boldsymbol{x} \in \mathcal{G}(\Omega_{k'}^{h,initial}, s)} \phi_{k'}^{h,initial}(\boldsymbol{x}) + \mathcal{I}(\phi^{H,corr}) \tag{30}$$

where $\mathcal{I}$ is the same interpolation operator used in step 2c of the single-grid infinite-domain Poisson solver (setting the layer width P to b), and

$$\phi^{H,corr} = \phi^H(\boldsymbol{x}) - \sum_{k' : \boldsymbol{x} \in \mathcal{G}(\Omega_{k'}^h, s)} \phi_{k'}^{H,initial}(\boldsymbol{x}). \tag{31}$$

Figure 2 depicts the regions from which data are taken to set boundary conditions on a face.

Finally, we apply the Mehrstellen correction (15) to the solution.
For $O(h^2)$ accuracy of the method, we set $b = 2$ and $s = 2C$.

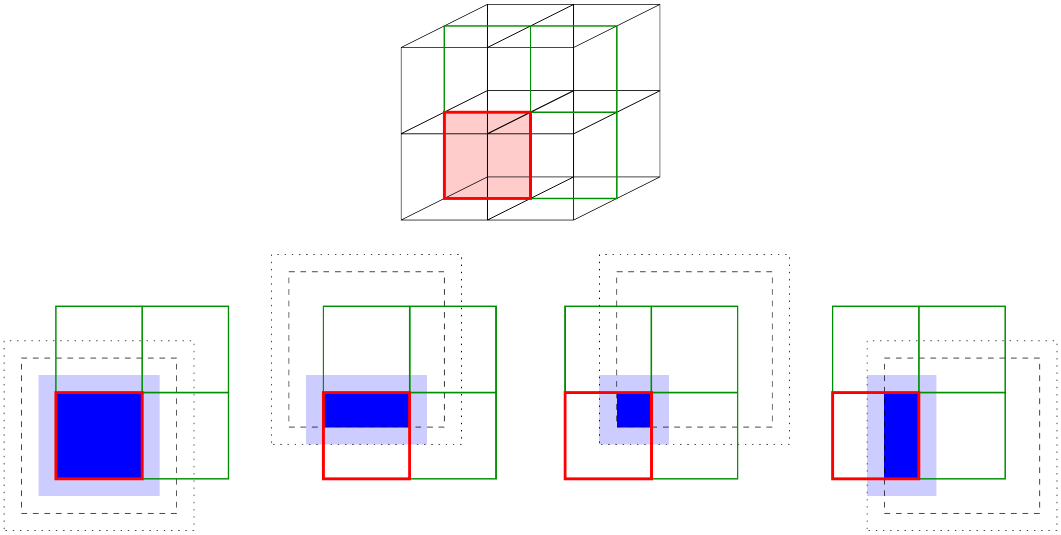

Figure 2: Setting boundary values for a final local solution in step 3 of MLC. For the face shaded red in the top of the figure, in a layout of eight cubes, the lower diagrams depict the regions from which data are copied from faces of different neighboring boxes. Solid lines indicate the boundaries of the boxes $\Omega^h_{k'}$, dashed lines the boundaries of the boxes $\mathcal{G}(\Omega^h_{k'}, s)$, and dotted lines the boundaries of the boxes $\mathcal{G}(\Omega^h_{k'}, s + Cb)$. Fine-grid data are copied to the red face from the nodes inside and on the edges of the regions shaded dark blue. Coarse grid data are copied from nodes inside and on the edges of the regions shaded both dark and light blue, and then interpolated to nodes on the red face that are inside and on the edges of the regions shaded dark blue.

4.1 Separating the Monopole Component

In order to minimize the cost of the infinite-domain solution, we would like to set s_1, the amount by which we grow the domain in the initial Dirichlet solution for the James-Lackner algorithm, to be zero. In the present application, the charge on each patch is nonzero all the way out to the boundary, so that the conditions under which this would be valid do not hold. In particular, for the fixed-size patches (relative to the mesh spacing) we are using here, this leads to an $O(1)$ relative error in the monopole component of the field used to compute the boundary conditions for the second Dirichlet solution. We eliminate this error by separating out the monopole component on each patch, and treating it exactly.

Specifically, for a given patch B, we compute $\bar{\rho}$, the mean of ρ over B, and to subtract $\bar{\rho}\chi^h_B$ from the right-hand side of (1) before solving, then to add $\bar{\rho}\xi_B$ to the solution, where $\xi_B \equiv G * \chi_B$ is computed analytically and stored.

In the initial local solve, we replace (24) by

$$\Delta^h \widetilde{\phi}_k^{h,initial} = \rho_k^h - \bar{\rho}_k^h \chi^h_{\Omega_k^h} \text{ on } \mathcal{G}(\Omega_k^h, s + Cb) \tag{32}$$

and then sample the solution:

$$\widetilde{\phi}_k^{H,initial} = \mathcal{S}^H(\widetilde{\phi}_k^{h,initial}) \text{ on } \mathcal{G}(\Omega_k^H, s/C + b). \tag{33}$$

The updated solutions are

$$\phi_k^{h,initial} = \widetilde{\phi}_k^{h,initial} + \bar{\rho}_k^h \xi^h_{\Omega_k^h}; \tag{34}$$

$$\phi_k^{H,initial} = \widetilde{\phi}_k^{H,initial} + \bar{\rho}_k^h \xi^H_{\Omega_k^H}. \tag{35}$$

In forming the right-hand side for the global coarse solve, we replace (26) by

$$R_k^H = \begin{cases} \Delta^H \widetilde{\phi}_k^{H,initial} + \bar{\rho}_k^h \chi^H_{\Omega_k^H} & \text{on } \mathcal{G}(\Omega_k^H, s/C - 1); \\ 0 & \text{elsewhere.} \end{cases}$$

In the final local Dirichlet solves, we replace (29) by

$$\Delta^h \widetilde{\phi}_k^h = \rho_k^h - \bar{\rho}_k^h \chi^h_{\Omega_k^h} \text{ on } \Omega_k^h \tag{36}$$

and the Dirichlet boundary conditions (30) by

$$\widetilde{\phi}_k^h(\boldsymbol{x}) = \sum_{k' : \boldsymbol{x} \in \mathcal{G}(\Omega_{k'}^{h,initial}, s)} \phi_{k'}^{h,initial}(\boldsymbol{x}) + \mathcal{I}(\phi^{H,corr}) + \bar{\rho}_k^h \xi^h_{\Omega_k^h}(\boldsymbol{x}). \tag{37}$$

Finally, we have the solution

$$\phi_k^h = \widetilde{\phi}_k^h + \bar{\rho}_k^h \xi^h_{\Omega_k^h}. \tag{38}$$

4.2 Extending to more than two levels

In extending the MLC solver from two levels to three, we assume a hierarchical nesting of patches, such that each fine-level patch is contained in one and only one middle-level patch. We run the two-level MLC solver separately within each middle-level patch, except that we perform the global coarse solve (28) on the middle-level patches with a two-level MLC solver using all the middle-level patches and the domain of the coarsest level.

The MLC solvers between the middle and fine levels require an expansion of the middle-level patches, specifically by $2\lceil (s/C-1)/2 \rceil$, taking s and C to be respectively the correction radius and coarsening ratio between these two levels. Then in the MLC solver between the middle and coarse levels, the finer-level patches Ω_k^h will overlap by this amount.

We may similarly extend to an arbitrary number of levels. In our implementation of the MLC solver on three levels, in order to retain accuracy we set $b = 2$ and a larger buffer size $s = 4C$ (instead of $s = 2C$) relating the coarse and the middle levels where C is the coarsening factor between middle and coarse levels. Between the fine and middle levels, we retain buffer size $s = 2C$ where C is the coarsening factor between these levels.

5 Results

As an example, we use right-hand sides built from ρ_m^{osc}, a spherically symmetric function with high-wavenumber component:

$$\rho_m^{osc}(r) \;=\; \begin{cases} ((r-r^2)\sin(2m\pi r))^2, & \text{if } r < 1\ ; \\ 0, & \text{if } r \geq 1\ . \end{cases}$$

The wavelength of ρ_m^{osc} is $\frac{1}{2m}$. If we set $\alpha = 4m\pi$, then the integral of ρ_m^{osc} over space is

$$\int \rho_m^{osc} dV = \pi\left(\frac{2}{105} + \frac{48}{\alpha^4} - \frac{1440}{\alpha^6}\right),$$

and the exact solution of

$$\Delta\phi_m^{osc} = \rho_m^{osc}$$

with infinite-domain boundary conditions is:

$$\phi_m^{osc}(r) \;=\; \begin{cases} r^6/84 - r^5/30 + r^4/40 + \\ 60/\alpha^6 - 9/\alpha^4 - 1/120 + 120/(\alpha^6 r) + \\ (-120/(\alpha^6 r) - 9/\alpha^4 + 300/\alpha^6 + 36r/\alpha^4 + r^2/(2\alpha^2) \\ -30r^2/\alpha^4 - r^3/\alpha^2 + r^4/(2\alpha^2))\cos(\alpha r) + \\ (12/(\alpha^5 r) - 360/(\alpha^7 r) - 96/\alpha^5 + 120r/\alpha^5 \\ -3r/\alpha^3 + 8r^2/\alpha^3 - 5r^3/\alpha^3)\sin(\alpha r), & \text{if } r < 1\ ; \\ (-1/210 - 12/\alpha^4 + 360/\alpha^6)/r, & \text{if } r \geq 1\ . \end{cases}$$

This solution is negative and has its minimum value at the origin:

$$\phi_m^{osc}(0) = -\frac{1}{120} - \frac{6}{\alpha^4}.$$

We test with three different charge densities on the unit cube $[0, 1]^3$, with m set to either 7, 15, or 30, and $R = 0.05$ in

$$\rho(\boldsymbol{x}) = \frac{1}{R^3}(\rho_m^{osc}(\frac{|\boldsymbol{x} - \boldsymbol{c}_1|}{R}) + \rho_m^{osc}(\frac{|\boldsymbol{x} - \boldsymbol{c}_2|}{R}) + \rho_m^{osc}(\frac{|\boldsymbol{x} - \boldsymbol{c}_3|}{R})), \tag{39}$$

where $\boldsymbol{c}_1 = (\frac{3}{16}, \frac{7}{16}, \frac{13}{16})$, $\boldsymbol{c}_2 = (\frac{7}{16}, \frac{13}{16}, \frac{3}{16})$, and $\boldsymbol{c}_3 = (\frac{13}{16}, \frac{3}{16}, \frac{7}{16})$. This is a superposition of three disjoint spherical charge distributions. The wavelength is $\lambda = R/(2m) = 1/(40m)$. The solution, which is negative, attains its minimum value at the sphere centers,

$$\phi^{exact}(\boldsymbol{c}_1) = \phi^{exact}(\boldsymbol{c}_2) = \phi^{exact}(\boldsymbol{c}_3) = (-\frac{1}{120} - \frac{6}{\alpha^4})/R + (-\frac{1}{105} - \frac{24}{\alpha^4} + \frac{720}{\alpha^6})/D$$

where $D = |\boldsymbol{c}_1 - \boldsymbol{c}_2| = |\boldsymbol{c}_1 - \boldsymbol{c}_3| = |\boldsymbol{c}_2 - \boldsymbol{c}_3|$ is the distance between any two sphere centers.

Our example uses three levels of boxes shown in Figure 3, with a coarsening factor of 4 between adjacent levels. The boxes are as follows.

- Fine level: all boxes are cubes of length 32. If the whole domain is split into $512 = 8^3$ subdomains of length $\frac{1}{8}$, then three of these subdomains contain the support of the charge; these subdomains are then fully refined with fine-level boxes.
- Middle level: all boxes are cubes of length 32 (becoming 36 after expansion, as described in section 4.2). Boxes at this level cover the three subdomains with the support of the charge, plus an additional layer of boxes.
- Coarse level: these boxes cover the entire domain and are parallel slabs in one direction. The number of slabs is the domain length in coarse cells divided by 4, or the number of processors, whichever is less.

5.1 Convergence results

In reporting our convergence results, we show the max norms and L^2 norms of solution error (difference between calculated solution and exact solution) normalized by the max norm of the exact solution. We also show the L^2 norm of the error on the finest grids alone. (For all the cases discussed here, the max norm on the finest grids is equal to the max norm on the whole domain.) We also calculate a convergence rate, p, defined such that if ϵ_f and ϵ_c are the norms of the solution error with mesh spacings h_f and h_c, respectively, then

$$p = \log_2(\frac{\epsilon_f}{\epsilon_c}) / \log_2(\frac{h_f}{h_c}). \tag{40}$$

See Table 1 for convergence results with m set to 7, 15, and 30, in the example with three-level MLC separating monopole solutions. The tables show the fine-level mesh spacing h and norms of the normalized solution error ϵ^h, which is the difference between calculated solution and exact solution, divided by $||\phi^{exact}||_\infty$, when the finest-level mesh spacing is h. While overall the solution error is $O(h^2)$, there is considerable variation in the rates, depending on the norm used and the grid resolution. This variation is not surprising, given the fact that there are multiple parameter choices

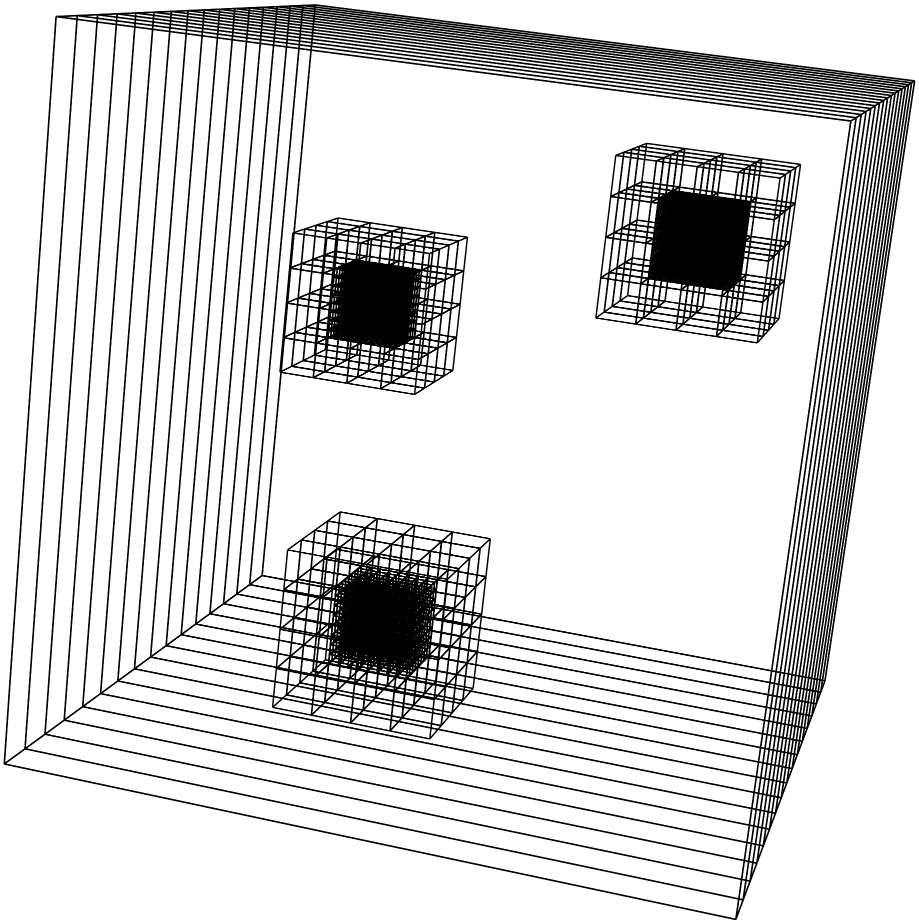

Figure 3: Boxes in three-level solve used in example. All boxes at the fine and middle levels have dimensions $32 \times 32 \times 32$. The coarse level is split into slabs across processors.

for the method that correspond to different asymptotic contributions to the error. The local James–Lackner computations have a contribution to the error that is $O(h^2)$ coming from the choice of multipole parameters, while the local truncation error for the Mehrstellen operator is $O(h^4)$, since we are applying the Mehrstellen correction in the form of (15). Finally, the choice of $b = 2$ in the boundary interpolation (30) for the final local solution step (29) corresponds to an error that is formally $O(h^6)$, although in this case, the contribution to the solution that is being interpolated is not sufficiently smooth to justify such an error estimate. In fact, the choice $b = 2$ was made empirically, with that choice leading to the most uniform convergence behavior. Such empirical choices are a weakness in the in the algorithm, and one that we intend to correct in future work.

Table 2 shows solution error on the three-level example with $m = 7$ when it is run *without* separating the monopole solution. This does not converge in L^2 norm, and has very poor convergence in max norm, thus illustrating the need for separating the monopole contribution to the solution.

Table 3 shows convergence results for the same examples (m set to 7, 15, and 30) with boxes at only two levels of refinement instead of three. The boxes at the fine level are the same as in the three-level arrangement, but the middle level is removed, and there is a coarsening factor of 4 from the fine level to a coarse level covering the full domain and split into parallel slabs. The mesh spacing at the coarser level in the two-level arrangement is the same as that of the middle level in the three-level arrangement. Comparing the two-level results in Table 3 and the three-level results in Table 1, with finest-level mesh spacings of $h = 1/2048$ and $h = 1/4096$, we see that the increased accuracy in the two-level calculation is worth approximately a factor of two in mesh spacing, with the difference decreasing as the wavenumber m increases. For the $m = 30$ cases, the three-level computation has essentially the same error as the two-level computation at the same fine-grid resolution. As m decreases to 15 and 7, the error of the two-level calculation becomes much smaller than that of the three-level calculation.This is consistent with the observation that there are two competing sources of error: that induced by the local truncation error, which for a fixed h scales like m^2; and that coming from the error in the representation of far-field effects, which is only weakly dependent on m. Thus as m decreases from $m = 30$, the contribution of the local truncation error rapidly decreases, leaving only the contribution from the error in the representation of the far-field effects. These are more accurately represented by a single-level calculation than by a two-level calculation at the same resolution. Nonetheless, we shall see below that, in these cases, the two-level and three-level calculations provide roughly the same accuracy for a given computational cost.

We also ran the problem on different sizes of a *single* grid with the James–Lackner solver of section 2 and Mehrstellen preconditioning (14). The results on the left side of Table 4 show solution error converging in max norm at a rate that is fourth order in the mesh spacing, as long as the oscillating right-hand side is resolved sufficiently. Nonetheless, the accuracy of the Mehrstellen method, by itself, is insufficient to make up for the lack of resolution in the coarsest-level calculation, so that the MLC method on the locally-refined grids substantially increases the accuracy of the overall solution. This is demonstrated in Table 4 by listing beside the Mehrstellen result, the max norm error of the three-level MLC result whose coarsest level has the same mesh spacing as that of the Mehrstellen result.

m	h	$\|\|\epsilon^h_{all}\|\|_\infty$	p	$\|\|\epsilon^h_{fine}\|\|_2$	p	$\|\|\epsilon^h_{all}\|\|_2$	p	λ/h
7	1/2048	2.132×10^{-5}		1.632×10^{-7}		1.738×10^{-7}		7.31
7	1/4096	4.735×10^{-6}	2.17	2.379×10^{-8}	2.78	4.712×10^{-8}	1.88	14.63
7	1/8192	1.130×10^{-6}	2.07	5.720×10^{-9}	2.06	8.419×10^{-9}	2.48	29.26
m	h	$\|\|\epsilon^h_{all}\|\|_\infty$	p	$\|\|\epsilon^h_{fine}\|\|_2$	p	$\|\|\epsilon^h_{all}\|\|_2$	p	λ/h
15	1/2048	2.437×10^{-5}		2.009×10^{-7}		2.357×10^{-7}		3.41
15	1/4096	4.906×10^{-6}	2.31	2.642×10^{-8}	2.93	3.061×10^{-8}	2.95	6.83
15	1/8192	1.157×10^{-6}	2.08	6.648×10^{-9}	1.99	9.737×10^{-9}	1.65	13.65
m	h	$\|\|\epsilon^h_{all}\|\|_\infty$	p	$\|\|\epsilon^h_{fine}\|\|_2$	p	$\|\|\epsilon^h_{all}\|\|_2$	p	λ/h
30	1/2048	5.022×10^{-5}		3.798×10^{-7}		3.848×10^{-7}		1.71
30	1/4096	5.274×10^{-6}	3.25	3.795×10^{-8}	3.32	6.296×10^{-8}	2.61	3.41
30	1/8192	1.542×10^{-6}	1.77	7.593×10^{-9}	2.32	1.270×10^{-8}	2.31	6.83

Table 1: Norms and convergence rates of solution error with adaptive three-level MLC separating monopole solutions, for example with $m = 7$, 15, and 30. The norms $||\epsilon^h_{fine}||$ are over the finest level, and $||\epsilon^h_{all}||$ are over all three levels.

m	h	$\|\|\epsilon^h_{all}\|\|_\infty$	p	$\|\|\epsilon^h_{fine}\|\|_2$	p	$\|\|\epsilon^h_{all}\|\|_2$	p	λ/h
7	1/2048	4.280×10^{-5}		8.449×10^{-7}		2.608×10^{-6}		7.31
7	1/4096	2.794×10^{-5}	0.62	7.009×10^{-7}	0.27	2.500×10^{-6}	0.06	14.63
7	1/8192	1.971×10^{-5}	0.50	6.713×10^{-7}	0.06	2.521×10^{-6}	-0.01	29.26

Table 2: Norms and convergence rates of solution error with adaptive three-level MLC *without* separating monopole solutions, for the example with $m = 7$. Compare with Table 1, which shows results of MLC separating monopole solutions. The norms $||\epsilon^h_{fine}||$ are over the finest level, and $||\epsilon^h_{all}||$ are over all three levels.

m	h	$\|\|\epsilon^h_{all}\|\|_\infty$	p	$\|\|\epsilon^h_{fine}\|\|_2$	p	$\|\|\epsilon^h_{all}\|\|_2$	p	λ/h
7	$1/2048$	4.498×10^{-6}		2.431×10^{-8}		2.471×10^{-8}		7.31
7	$1/4096$	9.698×10^{-7}	2.21	7.664×10^{-9}	1.67	3.373×10^{-8}	-0.45	14.63
15	$1/2048$	7.845×10^{-6}		7.889×10^{-8}		1.232×10^{-7}		3.41
15	$1/4096$	1.121×10^{-6}	2.81	6.919×10^{-9}	3.51	9.526×10^{-9}	3.69	6.83
30	$1/2048$	3.681×10^{-5}		3.380×10^{-7}		3.381×10^{-7}		1.71
30	$1/4096$	2.530×10^{-6}	3.86	2.365×10^{-8}	3.84	5.587×10^{-8}	2.60	3.41

Table 3: Norms and convergence rates of solution error with adaptive *two*-level MLC, for examples with $m = 7$, 15, and 30. Compare with Table 1, which shows results with three-level MLC. The norms $||\epsilon^h_{fine}||$ are over the finer level, and $||\epsilon^h_{all}||$ are over both levels.

	one-grid Mehrstellen				three-level MLC			
m	H	$\|\|\epsilon^H\|\|_\infty$	p	λ/H	h	$\|\|\epsilon^h\|\|_\infty$	p	λ/h
7	$1/256$	3.529×10^{-2}		0.91	$1/4096$	4.735×10^{-6}		14.63
7	$1/512$	4.193×10^{-4}	6.40	1.83	$1/8192$	1.130×10^{-6}	2.07	29.26
7	$1/1024$	1.726×10^{-5}	4.60	3.66				
15	$1/256$	1.019×10^{-2}		0.43	$1/4096$	4.906×10^{-6}		6.83
15	$1/512$	3.288×10^{-3}	1.63	0.85	$1/8192$	1.157×10^{-6}	2.08	13.65
15	$1/1024$	1.446×10^{-4}	4.51	1.71				
30	$1/256$	4.556×10^{-2}		0.21	$1/4096$	5.274×10^{-6}		3.41
30	$1/512$	4.167×10^{-3}	3.45	0.43	$1/8192$	1.542×10^{-6}	1.77	6.83
30	$1/1024$	9.687×10^{-4}	2.10	0.85				

Table 4: Max norms and convergence rates of solution error with Mehrstellen on a single grid (on left), for examples with $m = 7$, 15, and 30. Also shown (on right) are the max norms of the solution error for the three-level MLC, copied from Table 1, with the same mesh spacing H at the coarse level.

5.2 Timing results

In this section we present computational results demonstrating the low communication overhead of our implementation of the MLC algorithm on up to 1024 processors.

We ran on NERSC's Seaborg IBM SP system, located at the National Energy Research Scientific Computing Center[1]. Seaborg contains POWER3 SMP High Nodes interconnected with a "Colony" switch. Each node is an 16-way Symmetric Multiprocessor (SMP) based on 375 MHz Power-3 processors[2], sharing between 16 and 64 Gigabytes of memory, and running AIX version 5.1.

The solver is written in a mixture of C++ and Fortran 77, and calls the FFTW library [10] for the fast discrete sine transforms in the Dirichlet Poisson solves. We used the IBM C++ and Fortran 77 compilers, `mpCC` and `mpxlf`. C++ code was compiled with the IBM `mpCC` compiler, using options `-O2 -qarch=pwr3 -qtune=pwr3`. Fortran 77 was compiled with `mpxlf` with `-O2` optimization. We used the standard environment variable settings, and we collected timings in batch mode using *loadleveler*. The timings reported are based on wall-clock times, obtained with `MPI_Wtime()`.

The times reported are for the runs with the shortest total times of m set to 7, 15, or 30. Timers were placed around large function calls rather than inner loops to reduce the effects of noise in the timing results. The bulk-synchronous nature of the algorithm allows us to fully separate computation times from communication times. Reported running times do not include one-time startup costs such as a preprocessing phase for the serial James–Lackner solver that computes a matrix for obtaining outer-grid boundary conditions from multipole coefficients due to charges on the inner-grid boundary. This matrix depends only on the problem size and accuracy parameters, and its computation is considered a fixed overhead to be amortized over many calls to the solver.

In measuring the performance, we scaled the work with the number of processors. The run parameters and timing results for the performance tests of the three-level MLC are shown in Table 6. Processors are allocated to SMP nodes in such a way that each node runs 16 processors.

Results for performance tests of the two-level MLC are shown in Table 7. Since execution slows down when the memory capacity of a node is close to being reached, in the two-level MLC runs, processors are allocated to SMP nodes in such a way that each SMP node runs only eight processors — that is, half of the processors on the node.

Results for performance tests of the parallelized single-grid solver are shown in Table 8. In these examples, as with the two-level MLC runs, processors are allocated to SMP nodes with eight processors per node.

We define grind time as the processor-time taken per fine-level solution point. Ideally the grind time would remain constant over problem sizes and numbers of processors. We see from Table 6 that for the three-level MLC, grind times are fairly stable, at around 22 to 23 μsec/point.

Grind times vary by almost a factor of two in the two-level MLC examples in Table 7. These results are not as consistent as with the three-level example, because the coarse-level solution takes a majority of the run time: it is computed using a conventional parallel FFT algorithm that does not scale as well as the local solves, and has 2.65 times as many points as the local fine grids (Table 5). This lack of scaling also had an impact on the memory requirements. The two-level calculations required substantially more memory than the three-level calculations, so that we were only able

---

[1] http://www.nersc.gov/nusers/resources/SP

[2] http://www-1.ibm.com/servers/eserver/pseries/hardware/whitepapers/nighthawk.html

Size	three-level example			two-level example	
N	fine	middle	coarse	fine	coarse
2048	50 923 779	6 440 067	2 146 689	50 923 779	135 005 697
4096	405 017 091	21 567 171	16 974 593	405 017 091	1 076 890 625
8192	3 230 671 875	99 228 483	135 005 697		

Table 5: Numbers of solution points at each level in the three-level MLC example (with timing results in Table 6) and the two-level MLC example (with timing results in Table 7).

	Size	Times for each stage (seconds)							Total	Grind
P	N	InitF	InitM	Crse	BndM	FinM	BndF	FinF	(sec)	(μsec/pt)
16	2048	44.99	12.52	3.51	0.33	0.66	2.64	4.89	69.57	21.86
128	4096	45.51	6.76	10.19	0.15	0.30	4.12	4.75	71.83	22.70
1024	8192	46.01	3.95	13.04	0.15	0.17	4.03	4.78	72.28	22.91

Table 6: Timing breakdowns for runs of an adaptive three-level MLC solver, with P processors and domain length N. Boxes at the fine and middle levels are all cubes of length 32. InitF: Initial fine-level local solve. InitM: Initial middle-level local solve. Crse: Coarse-level solve. BndM: Boundary communication to middle-level final solve. FinM: Final middle-level solve. BndF: Boundary communication to fine-level final solve. FinF: Final fine-level solve.

	Size	Times for each stage (seconds)				Total	Grind
P	N	Init	Crse	Bnd	Fin	(sec)	(μsec/pt)
64	2048	11.73	22.61	0.26	1.22	35.84	45.04
512	4096	13.25	47.46	0.50	1.21	62.44	78.93

Table 7: Timing breakdowns for runs of an adaptive two-level MLC solver, with P processors and domain length N. Boxes at the fine level are all cubes of length 32. Init: Initial fine-level local solve. Crse: Coarse-level solve. Bnd: Boundary communication to fine-level final solve. Fin: Final fine-level solve.

Size			Times for each stage (seconds)				Total	Grind
P	N	points	Homo	Normal	FMM	Inhomo	(sec)	(μsec/pt)
4	256	16 974 593	10.53	0.08	2.23	57.34	70.20	16.54
32	512	135 005 697	13.39	0.87	4.51	22.93	41.72	9.89
256	1024	1 076 890 625	13.65	3.06	10.53	19.26	46.52	11.06

Table 8: Timing breakdowns for runs of infinite-domain solver on one level, with P processors and domain length N, and given number of points, which is $(N+1)^3$. Homo: Initial homogeneous Dirichlet Poisson solve. Normal: Copying of Poisson solution and evaluation of normal derivatives. FMM: Fast multipole method. Inhomo: Final inhomogeneous Dirichlet Poisson solve.

P	Size N	Communication in stages (seconds) Boundary	Coarse	Residuals	Total (sec)	% of runtime
16	2048	0.37	0.22	0.08	0.68	0.97 %
128	4096	1.56	0.58	0.14	2.28	3.17 %
1024	8192	1.40	1.77	0.68	3.85	5.32 %

Table 9: Communication time in the adaptive three-level MLC solve, for the same runs as reported in Table 6. Boundary: Copying of solutions within and between fine and middle levels (as illustrated in Figure 2). Coarse: Communication in the solve at the coarsest level. Residuals: Copying of residuals at the fine and middle levels.

to use eight processors per node for these runs, rather than the full 16 processors per node used in the three-level runs. In reporting the number of processors and computing the grind times in Table 7, we report the number of processors actually used, whereas the system resources required corresponded to double that number.

The lower parallel performance of the two-level calculations also affects the tradeoffs between using the two-level and three-level algorithms from an accuracy standpoint. For the $m = 30$ case, the accuracy of the two-level and three-level calculations are almost the same, and the cost of the two-level calculation is far greater: for example, the system resources required for the 4096-resolution two-level calculation are the same as those used for the 8192-resolution three-level calculation. As m decreases, the tradeoffs favor the two-level calculation more, but the computational costs of obtaining a given level of accuracy using the two different strategies remains within a factor of two.

In the runs of the three-level MLC, as shown in Table 6, over half the time is spent in the initial fine-level solves. With the particular problem sizes, each processor holds data for 96 fine-level boxes. The grind time for the initial fine-level solves ranges from 14.1 to 14.6 μsec/point, and for the final fine-level solves ranges from 1.50 to 1.54 μsec/point. Overall, we are able to scale a problem up from 16 to 1024 processors with, at worst, a 4% increase in the grind time. The increase is due primarily to the increased cost of the global FFT solution at the coarsest level. We believe that the performance of our implementation of the global FFT solver can be improved from that seen here.

We compare these results with timings for an adaptive node-centered multigrid algorithm for solving Poisson's equation with Dirichlet boundary conditions [18] on the same platform. This algorithm is run on three levels of boxes, with the fine and middle levels being the same as were used with MLC, but the coarse level being fully refined into cubes of length 32, instead of parallel slabs. Although we are solving a different problem here, we believe that these results are typical of the cost of of using the same algorithm to solving the infinite-domain problem along the lines of the algorithm in [1]. Comparison of results of the multigrid timings in Table 10 with the MLC timings in Table 6 shows that the multigrid algorithm takes 5 to 7 times longer than MLC, although a count of the number of floating-point operations shows that it uses only 1.38 to 1.45 times as many such operations as MLC. Considerably more time is spent in communication in this algorithm than in MLC (comparing Table 10 with Table 9). On the example on 128 processors, the time for communication, at 78.87 seconds, exceeds the total time for the MLC solve on the same grids.

P	Size N	AMR iterations	Time (sec)	Grind (μsec/pt)	Communication time (sec)	 % of runtime
16	2048	9	352.04	110.61	25.81	7.49%
128	4096	9	468.41	148.03	78.87	17.65%

Table 10: Times for multigrid Poisson solver with Dirichlet boundary conditions. Compare with Table 6 for MLC on the same fine-level and middle-level grids.

Finally, we can infer from the results given here a lower bound on the grind time required to use a Hockney algorithm for solving Poisson's equation at the same resolution as that on our finest grid. We can infer the time per grid point of an FFT solver for a 32^3 grid from the time given for the final fine grid solution in Table 6 to be about 1.52 μsec per grid point. Thus the cost per mesh point per processor of performing an infinite-domain solution on a uniform grid with linear dimension N using the James-Lackner algorithm is at least $1.52 \times 2 \times .2log_2(N)\mu$sec, where the factor of 2 comes from the minimum cost of solving the two Dirichlet problems for the James-Lackner algorithm, and $.2 = 1/log_2(32)$. This leads to grind times of 6.7 - 7.9 μsec for the range of mesh resolutions given here. Thus, the grind times for the three-level MLC calculations are approximately a factor of three times the lower bound we've estimated here.

6 Conclusions and Future Work

We have described here an extension to Anderson's Method of Local Corrections for solving Poisson's equation in free space on nested multiresolution grids in three dimensions. This is a non-iterative domain-decomposition method based on computing local convolutions with the free-space Green's function on overlapping rectangular subdomains with a fixed number of grid points, combined with a representation of the nonlocal coupling between subdomains by a coarse-grid calculation in a manner that is structurally similar to a single iteration of an FAS multigrid method. The extension to locally-refined grids, and to more than two levels is straightforward. A key technical step is an extension to three dimensions of the James–Lackner method for computing local convolutions, based on using FFTs for computing the volume potentials combined with a simplified version of the fast multipole method for surface-surface convolutions. This is combined with an exact treatment of the contribution to the local potential from the piecewise-constant component of the charge in each rectangle.

We demonstrated second-order accuracy of the method for a nontrivial example. We also found that the computational cost of the method is approximately three times per grid point that of FFT calculation at the same resolution, and scales to 1024 processors at approximately 95% parallel efficiency, with less than 7% of the run time in MPI communication calls. We have also compared the performance of this method to that of a conventional AMR multigrid solver on the same grid hierarchy, and found that, on 128 processors, the latter takes seven times as much time overall to compute the result, and spends 16 times as much time in MPI communication than the present method. We know of no other method for Poisson's equation in 3D that exhibits the same combination of performance and scalability on multiresolution grid. We believe that the results presented here indicate the possibility of scaling effectively to a PetaFlop computer (10^5 processors).

The results given here, while extremely promising, must be viewed as a first step in developing a robust and automatic piece of software. There are free parameters in the method, such as the dependence of the degree of overlap on the level of refinement, that are ad-hoc, and need to be defined systematically. One of the principal difficulties in this area is estimating and controlling the different sources of error in the algorithm separately and with complete generality. One aspect of solving that problem is for all components of the algorithm to have tunable accuracy, as opposed the present situation, in which the fourth-order Mehrstellen algorithm is a fixed target. Also, the current formulation of the algorithm does not preserve the geometric locality of the charge distributions. For example, the field induced on a patch on the middle level in a three-level calculation includes contributions from the charge distribution on finer patches not covered by the middle patch. This feature makes it difficult to estimate the error as it propagates down through refinement levels. We are currently working on a version of the algorithm that will preserve locality under coarsening. As is the case with other adaptive methods, the general question of criteria for determining the needed grid refinement, as a function of space, time, and data, is not completely resolved. However, our treatment of the coupling between refinement levels may simplify the problem, relative to conventional finite difference methods [18]. Finally, there is still room for further performance improvement. For example, the parallel FFT solver used for the coarsest level is implemented using the Chombo communication primitives, that were not designed for the global communications required in the transform step. Since this calculation is a parallel bottleneck for the overall algorithm, any improvements would significantly improve the overall scaling of the method.

There are a number of directions in which the method described here could be extended. These include cell-centered solvers, solvers for the 3D Helmholtz equations, and higher-order methods. The extension to other boundary conditions on the domain boundary (Dirichlet, Neumann) is straightforward using method of images ideas; a more challenging question is the extension of this approach to the case of Cartesian-grid representations of irregular boundaries [19, 15, 18].

7 Acknowledgments

Peter McCorquodale and Phillip Colella are supported by the Mathematical, Information, and Computational Sciences Division of the Office of Science, U.S. Department of Energy under contract number DE-AC03-76SF00098. Gregory Balls and Scott Baden were supported by the National Partnership for Advanced Computational Infrastructure (NPACI) under NSF contract ACI9619020. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

References

[1] A. S. Almgren, T. Buttke, and P. Colella. A fast adaptive vortex method in 3 dimensions. *J. Comput. Phys.*, 113(2):177–200, August 1994.

[2] C. R. Anderson. A method of local corrections for computing the velocity field due to a distribution of vortex blobs. *J. Comput. Phys.*, 62:111–123, 1986.

[3] Scott B. Baden. *Run-time Partitioning of Scientific Continuum Calculations Running on MultiProcessors*. PhD thesis, UC Berkeley Computer Science Division, April 1987.

[4] D. Bai and A. Brandt. Local mesh refinement multi-level techniques. *SIAM Journal Sci.Stat. Comput.*, 8:109–134, 1987.

[5] G. T. Balls. *A Finite Difference Domain Decomposition Method Using Local Corrections for the Solution of Poisson's Equation*. PhD thesis, University of California, Berkeley, 1999.

[6] G. T. Balls and P. Colella. A finite difference domain decomposition method using local corrections for the solution of Poisson's equation. *J. Comput. Phys.*, 180(1):25–53, July 2002.

[7] A. Brandt. Multilevel adaptive methods for boundary-value problems. *Math. Comput.*, 31(138):333–390, April 1977.

[8] H. Cheng, J. Huang, and T. J. Leiterman. A fast adaptive solver for the modified Helmholtz equation in two dimensions. *J. Comput. Phys.*, 211(2):616–637, January 2006.

[9] F. Ethridge and L. Greengard. A new fast-multipole accelerated Poisson solver in two dimensions. *SIAM Journal Sci. Comput.*, 23(3):741–760, 2001.

[10] M. Frigo and S. G. Johnson. FFTW: An adaptive software architecture for the FFT. In *ICASSP Conference Proceedings*, volume 3, pages 1381–1384. ICASSP, 1998.

[11] L. Greengard and J.-Y. Lee. A direct adaptive Poisson solver of arbitrary order accuracy. *J. Comput. Phys.*, 125(2):415–424, May 1996.

[12] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73:325–348, 1987.

[13] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. McGraw-Hill, 1981.

[14] R. A. James. The solution of Poisson's equation for isolated source distributions. *J. Comput. Phys.*, 25(2):71–93, October 1977.

[15] H. Johansen and P. Colella. A Cartesian grid embedded boundary method for Poisson's equation on irregular domains. *J. Comput. Phys.*, 147(2):60–85, December 1998.

[16] Oliver D. Kellogg. *Foundations of Potential Theory*. Dover Publications, 1953.

[17] K. Lackner. Computation of ideal MHD equilibria. *Computer Physics Communications*, 12(1):33–44, 1976.

[18] P. McCorquodale, P. Colella, D. Grote, and J.-L. Vay. A node-centered local refinement algorithm for Poisson's equation in complex geometries. *J. Comput. Phys.*, 201:34–60, 2004.

[19] G. H. Shortley and R. Weller. The numerical solution of Laplace's equation. *J. Appl. Phys.*, 9:334–348, 1938.

[20] Barry F. Smith, Petter E. Bjørstad, and William D. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 2004.